Take Everything From Me, But Leave Me The Comprehension

DBPL — September 2017

Torsten Grust

db.inf.uni-tuebingen.de
Apologies, I am only a database person
It is in this connection worth noticing that in the Comm.ACM the papers on data bases [...] are of markedly lower linguistic quality than the others.

—Edsger Dijkstra (EWD691)
The point is that the way in which the database management experts tackle the problems seems to be so grossly inadequate. They seem to form an inbred crowd with very little knowledge of computing science in general, who tackle their problems primarily politically instead of scientifically.

—Edsger Dijkstra (EWD577)
Apologies, I am only a database person

Often they seemed to be mentally trapped by the intricacies of early, rather ad hoc solutions to rather accidental problems; as soon as such a technique has received a name, it becomes "a database concept". And a totally inadequate use of language, sharpening their pencils with a blunt axe.

—Edsger Dijkstra (EWD577)
Apologies, I am only a database person

I learned a few things about Databases. I learned—or: had my tentative impression confirmed—that the term "Database Technology", although sometimes used, is immature, for there is hardly any underlying "science" that could justify the use of the term "technology".

—Edsger Dijkstra (EWD577)
Comprehension Syntax

\[ [ \ h x \ \ | \ \ x \leftarrow xs, \ p x \ ] \]
Comprehension Syntax

\[ \left[ h \ x \ | \ x \leftarrow xs, \ p \ x \right] \]

1. Successively **draw bindings** for \( x \) from domain \( xs \),
2. for those bindings that pass **filter** \( p \),
3. evaluate **head** \( h \),
Comprehension Syntax

\[ \{ h \, x \mid x \leftarrow xs, \ p \, x \} \]

1. Successively **draw bindings** for \( x \) from domain \( xs \),
2. for those bindings that pass **filter** \( p \),
3. evaluate **head** \( h \),
4. collect results to form a list.
Comprehension Syntax

\[ [ h \ x \mid \ x \leftarrow \ xs, \ p \ x ]^M \]

1. Successively **draw bindings** for \( x \) from domain \( xs \),
2. for those bindings that pass **filter** \( p \),
3. evaluate **head** \( h \),
4. collect results to form an \( M \).
Extension vs. Intension

{ } { }
Extension vs. Intension

$\{ I, III, V, VII, IX \}$
Extension vs. Intension

\[ \{ \text{I, III, V, VII, IX} \} \]

\[ \{ \text{roman } x \mid x \leftarrow [1\ldots10], \text{ odd } x \}^{\text{set}} \]
In the Beginning …

Relational Completeness of Data Base Sublanguages
E. F. Codd, IBM Research Report RJ987, 1972

\((r_1[3], r_2[2]): \phi_{r_1} \land \phi_{r_2} \land (r_1[1] = r_2[1]).\)
In the Beginning ...

987

RELATIONAL COMPLETENESS OF DATA BASE SUBLANGUAGES

by

E. F. Codd

IBM Research Laboratory
San Jose, California

Relational Completeness of Data Base Sublanguages
E. F. Codd, IBM Research Report RJ987, 1972

(generator)

$(r_1[3], r_2[2]): p_1 r_1 \land p_2 r_2 \land (r_1[1] = r_2[1])$. 

(head)

(filter)
Today’s XQuery 3.0

Core of XQuery: versatile FLWOR expression
Today’s XQuery 3.0

Core of XQuery: versatile FLWOR expression

sequence. The query returns one value

for $x$ at $i$ in $inputvalues$
where $i \mod 100 = 0$
return $x$

XQuery 3.0: An XML Query Language
D. Chamberlin et al., W3C Recommendation, April 2014
Today’s XQuery 3.0

Core of XQuery: versatile *FLWOR* expression
We differ from other presentations of nested relational algebra in that we make heavy use of list comprehensions, a standard notation in the functional programming community [1]. We find list comprehensions slightly easier to manipulate than the more traditional algebraic operators, but it is not hard to translate comprehensions into these operators (or vice versa).

We can use comprehensions to express fundamental query operations such as nesting, and joins.

We can navigate from a node to all of its children elements with the follow operator.

\[
\begin{align*}
\text{follow} & \quad ::= \text{Tag} \to \text{Node} \to [\text{Node}] \\
\text{follow t x} & \quad = \quad [ y \mid y \leftarrow \text{children x, is t y} ]
\end{align*}
\]
Early XQuery

We differ from other presentations of nested relational algebra in that we make heavy use of list comprehensions, a standard notation in the functional programming community [1]. We find list comprehensions slightly easier to manipulate than the more traditional algebraic operators, but it is not hard to translate comprehensions into these operators (or vice versa).

We can use comprehensions to express fundamental query operations such as nesting, and joins.

We can navigate from a node to its children elements with a generator:

\[
\text{follow} \quad :: \quad \text{Tag} \rightarrow \text{Node} \rightarrow \{ \text{Node} \}
\]

\[
\text{follow } t \ x \quad = \quad [ \ y \mid y \leftarrow \text{children } x, \text{is } t \ y \ ]
\]
An XQuery Nucleus

```xquery
1 module Query where
2 import Prelude hiding (elem,index)

4 -- Data Model: Constructors ---------------------------------------------

6 text :: String -> Node
7 elem :: Tag -> [Node] -> Node
8 ref :: Node -> Node

10 year0 :: Node
11 year0 = elem "@year" [ text "1999" ]

13 book0 :: Node
14 book0 = elem "book" [ elem "@year" [ text "1999" ],
15 elem "title" [ text "Data on the Web" ],
16 elem "author" [ text "Abiteboul" ],
17 elem "author" [ text "Buneman" ],
18 elem "author" [ text "Suciu" ] ]
```
An XQuery Nucleus

• ≈ 430 lines of Haskell (300+ lines of examples)
• Implements a complete XQuery core, including tree construction and traversal
• **List comprehensions** express path navigation, *FLOWR*, grouping/aggregation, quantification
Comprehension syntax deeply embedded into C#, with monad-based semantics organized around `SelectMany` (aka `>>=`, `flatMap`).
The LINQ comprehension syntax is just syntactic sugar for a set of standard query operators that can be defined in any modern programming language with lambda expressions (written here as `rating=>rating == "****"`), or

```csharp
var q = from product in Products
where product.Ratings.Any(rating=>rating == "****")
select new { product.Title, product.Keywords };
```

Comprehension syntax deeply embedded into C#, with monad-based semantics organized around `SelectMany` (aka `>>=`, `flatMap`).
Listing 6: Page Rank in Emma

```scala
var iter = 0
while (iter < maxIterations) {
    val messages = for {
        p <- ranks.bag();
        v <- vertices; n <- v.neighbors;
        if p.id == v.vertex) yield {
            RankMessage(n, p.rank / v.neighbors.count())
        }
    }
```
Deep embedding of comprehensions in Scala, compiles to Apache Flink / Spark
Pig Latin

Compiles to sequences of *Map/Reduce* jobs

```
X = FOREACH A GENERATE a1+a2 AS f1:int;
DESCRIPT X;
x: {f1: int}
DUMP X;
(3)
(6)
(11)
(7)
(7)
(9)
(12)
Y = FILTER X BY f1 > 10;
DUMP Y;
(11)
(12)
```
Pig Latin

Compiles to sequences of Map/Reduce jobs

```pig
X = FOREACH A GENERATE a1+a2 AS f1:int;

DESCRIBE X;
x: {f1: int}

DUMP X;
(3)
(6)
(11)
(7)
(9)
(12)

Y = FILTER X BY f1 > 10;
DUMP Y;
(11)
(12)
```
**Pig Latin**

Compiles to sequences of *Map/Reduce* jobs

---

Note: FOREACH statements can be nested to two levels only. FOREACH statements that are nested to three or more levels will result in a grammar error.
Note: FOREACH statements can be nested to two levels only. FOREACH statements that are nested to three or more levels will result in a grammar error.

X = FOREACH A GENERATE a1+a2 AS f1:int;
DESCRIBE X;
x: {f1: int}

Y = FILTER X BY f1 > 10;
DUMP Y;

SELECT o_orderpriority, COUNT(*) AS order_count
FROM orders
WHERE o_orderdate > @DATE@
AND o_orderdate < @DATE@ + interval '3 months'
AND EXISTS (SELECT *
              FROM lineitem
              WHERE l_orderkey = o_orderkey
              AND l_commitdate < l_receiptdate)
GROUP BY o_orderpriority
ORDER BY o_orderpriority;

Query Q4 of the TPC-H OLAP benchmark
```
SELECT o_orderpriority, COUNT(*) AS order_count
FROM orders
WHERE o_orderdate > @DATE@
AND o_orderdate < @DATE@ + interval '3 months'
AND EXISTS (SELECT *
    FROM lineitem
    WHERE l_orderkey = o_orderkey
    AND l_commitdate < l_receiptdate)
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
```

Query Q4 of the TPC-H OLAP benchmark
SELECT o_orderpriority, COUNT(*) AS order_count
FROM orders
WHERE o_orderdate > @DATE@
AND o_orderdate < @DATE@ + interval '3 months'
AND EXISTS (SELECT *
             FROM lineitem
             WHERE l_orderkey = o_orderkey
             AND l_commitdate < l_receiptdate)
GROUP BY o_orderpriority
ORDER BY o_orderpriority;
SELECT o_orderpriority, COUNT(*) AS order_count
FROM orders
WHERE o_orderdate > @DATE@
AND o_orderdate < @DATE@ + INTERVAL '3 months'
AND EXISTS (SELECT * 
FROM lineitem 
WHERE l_orderkey = o_orderkey
AND l_commitdate < l_receiptdate)
GROUP BY o_orderpriority
ORDER BY o_orderpriority;

Query Q4 of the TPC-H OLAP benchmark
SQL
One Way to Teach SQL
One Way to Teach SQL

```sql
SELECT A, B
FROM S
```
One Way to Teach SQL

```sql
SELECT A, B
FROM S
```

c ← ∅;

foreach x ∈ S do
  c ← c ∪ {(x.A, x.B)};

return c;

One Way to Teach SQL

```
SELECT A, B
FROM S
```

```
SELECT MAX(A)
FROM S
```

```
c ← ∅;
foreach x ∈ S do
  c ← c ∪ {(x.A, x.B)};
return c;
```
One Way to Teach SQL

```
SELECT A, B
FROM S
```

```
SELECT MAX(A)
FROM S
```

```
c ← ∅;
foreach x ∈ S do
c ← c ∪ {(x.A,x.B)};
return c;
```

```
c ← −∞;
foreach x ∈ S do
c ← max_2(c,x.A);
return c;
```
One Way to Teach SQL

\[
\begin{align*}
\text{SELECT} & \quad \text{A, B} \\
\text{FROM} & \quad \text{S}
\end{align*}
\]

\[
\begin{align*}
\text{c} & \leftarrow \emptyset; \\
\text{foreach } x & \in S \text{ do} \\
\quad \text{c} & \leftarrow \text{c} \cup \{(x.A,x.B)\}; \\
\text{return } c;
\end{align*}
\]

\[
\begin{align*}
\emptyset & \leftarrow -\infty; \\
\text{foreach } x & \in S \text{ do} \\
\quad \text{c} & \leftarrow \text{max}_2(c,x.A); \\
\text{return } c;
\end{align*}
\]

\[
\emptyset \leftarrow \text{ALL(}\text{SELECT} \quad \text{A} \\
\text{FROM} & \quad \text{S})
\]
One Way to Teach SQL

SELECT A, B
FROM S

SELECT MAX(A)
FROM S

\( \bigcirc \ll  \text{ALL} \left( \text{SELECT A} \right) \ll \bigcirc \text{FROM S} \)

c \leftarrow \emptyset ;
foreach x \in S do
c \leftarrow c \cup \{(x.A,x.B)\};
return c;

c \leftarrow -\infty ;
foreach x \in S do
c \leftarrow \max_2(c,x.A);
return c;

c \leftarrow \text{true} ;
foreach x \in S do
c \leftarrow c \land (\emptyset < x.A);
return c;
One Way to Teach SQL

```
SELECT A, B
FROM S

SELECT MAX(A)
FROM S

0 < ALL(SELECT A
FROM S)
```

c ← ∅;
for each x ∈ S do
c ← c ⨿ {(x.A,x.B)};
return c;

c ← −∞;
for each x ∈ S do
c ← max₂(c,x.A);
return c;

c ← true;
for each x ∈ S do
c ← c ∧ (0 < x.A);
return c;

One Program Form for SQL
One Program Form for SQL

\[
\text{fold}(z, f, xs) \equiv \\
c \leftarrow z; \\
\text{foreach } x \in xs \text{ do } \\
c \leftarrow f(c, x); \\
\text{return } c;
\]
One Program Form for SQL

\[ \text{fold}(z,f,xs) \equiv c \leftarrow z; \]
\[ \text{foreach } x \in xs \text{ do } c \leftarrow f(c,x); \]
\[ \text{return } c; \]

<table>
<thead>
<tr>
<th>( M )</th>
<th>carrier</th>
<th>( \text{lift}^M )</th>
<th>( z^M )</th>
<th>( \oplus^M )</th>
</tr>
</thead>
<tbody>
<tr>
<td>bag</td>
<td>bag ( t )</td>
<td>{ \cdot }</td>
<td>\emptyset</td>
<td>\cup</td>
</tr>
<tr>
<td>set</td>
<td>set ( t )</td>
<td>{ \cdot }</td>
<td>\emptyset</td>
<td>\cup</td>
</tr>
<tr>
<td>list</td>
<td>list ( t )</td>
<td>[.]</td>
<td>[]</td>
<td>\uplus</td>
</tr>
<tr>
<td>all</td>
<td>bool</td>
<td>id</td>
<td>true</td>
<td>\wedge</td>
</tr>
<tr>
<td>some</td>
<td>bool</td>
<td>id</td>
<td>false</td>
<td>\vee</td>
</tr>
<tr>
<td>sum</td>
<td>num</td>
<td>id</td>
<td>0</td>
<td>+</td>
</tr>
<tr>
<td>max</td>
<td>( t ) (ordered)</td>
<td>id</td>
<td>( -\infty )</td>
<td>\max_2</td>
</tr>
<tr>
<td>min</td>
<td>( t ) (ordered)</td>
<td>id</td>
<td>( \infty )</td>
<td>\min_2</td>
</tr>
</tbody>
</table>
One Program Form for SQL
One Program Form for SQL

\[
\text{SELECT } A \\
\text{FROM } S \\
\text{WHERE } A > B
\]

fold(∅, ⊕, S) with

⊕(c, x) = c ⊔ (if (x.A > x.B) {x.A} else ∅)
One Program Form for SQL

\[
\text{SELECT } A \\
\text{FROM } S \\
\text{WHERE } A > B
\]

fold(∅, ⊕, S) with
\[
⊕(c, x) = c \cup (\text{if } (x.A > x.B) \{x.A\} \text{ else } ∅)
\]

\[
\text{SELECT } x.A, y.B \\
\text{FROM } R x, S y
\]

fold(∅, ⊕, R) with
\[
⊕(c, x) = c \cup \text{fold}(∅, ⊗, S) \text{ with } \forall (d, y) = d \cup \{(x.A, y.B)\}
\]
fold(,,,)

Gets Ugly Quickly
fold(,,) Gets Ugly Quickly

```sql
SELECT COUNT(*)
FROM R x
WHERE EXISTS (SELECT y
               FROM S y
               WHERE x.A = y.B)
```

fold(0,⊕,fold(∅,⊗,R)) with
with ⊕(c,_) = c + 1
⊗(d,x) = d ∪ if (fold(false,⊙,S)) {x} else ∅
with ⊙(e,y) = e ∨ (x.A = y.B)
fold(,,,) Gets Ugly Quickly

\[
\begin{align*}
\text{SELECT} & \quad \text{COUNT}(*) \\
\text{FROM} & \quad R \times \\
\text{WHERE} & \quad \exists (\text{SELECT} y \\
& \quad \text{FROM} \quad S \ y \\
& \quad \text{WHERE} \quad x.A = y.B) \\
\end{align*}
\]

\[
\begin{align*}
\text{fold}(0, \oplus, \text{fold}(\emptyset, \otimes, R)) \\
\text{with } \oplus(c, \_ ) & = c + 1 \\
\otimes(d, x) & = d \uplus \text{if } (\text{fold}(\text{value}(x), d, S)) \neq \emptyset \\
\text{with } \otimes(e, y) & = e \lor (x.A = y.B)
\end{align*}
\]

ALGEBRAIC WONDERLAND.
fold(,,, ) Gets Ugly Quickly

SELECT COUNT(*)
FROM R x
WHERE EXISTS (SELECT y
FROM S y
WHERE x.A = y.B)

fold(0, ⊕, fold(∅, ⊗, R))
with ⊕(c, _) = c + 1
⊗(d, x) = d ∪ if (fold(∅, ⊗, S))
with ⊗(e, y) = e ∨ (x.A = y.B)

ALGEBRAIC WONDERLAND.
REJECT!
Comprehension Semantics
Comprehension Semantics

\[
\begin{align*}
[ e & ]^M \\
[ e & v_1 \leftarrow e_1, q ]^M \\
[ e & p, q ]^M
\end{align*}
\]
Comprehension Semantics

\[ [ \ e \ | \ ]^M \equiv \ \text{lift}^M(e) \]

\[ [ \ e \ | \ v_1 \leftarrow e_1, q \ ]^M \equiv \]

\[ [ \ e \ | \ p, q \ ]^M \equiv \]
Comprehension Semantics

\[ [ e \mid ]^M \equiv \text{lift}^M(e) \]

\[ [ e \mid v_1 \leftarrow e_1, q ]^M \equiv \text{fold}(z^M, \otimes, e_1) \text{ with } \otimes(c, v_1) = c \oplus^M [ e \mid q ]^M \]

\[ [ e \mid p, q ]^M \equiv \text{if} (p) [ e \mid q ]^M \text{ else } z^M \]
Comprehensible SQL
SELECT COUNT(*)
FROM R x
WHERE EXISTS (SELECT y
              FROM S y
              WHERE x.A = y.B)
Comprehensible SQL

```
SELECT COUNT(*)
FROM R x
WHERE EXISTS (SELECT y
              FROM S y
              WHERE x.A = y.B)
```

\[
[ y | y \leftarrow S, \ x.A = y.B ]^{bag}
\]
Comprehensible SQL

```sql
SELECT COUNT(*)
FROM R x
WHERE EXISTS (SELECT y
               FROM S y
               WHERE x.A = y.B)

[ true | _ ← [ y | y ← S, x.A = y.B ]^{bag} ]^{some}
```
Comprehensible SQL

```
SELECT COUNT(*)
FROM R x
WHERE EXISTS (SELECT y
  FROM S y
  WHERE x.A = y.B)
```

```
[ 1 | x ← R,
  [ true | _ ← [ y | y ← S, x.A = y.B ]^{bag} ]^{some} ]^{sum}
```
Comprehensible SQL

```
SELECT COUNT(*)
FROM R x
WHERE EXISTS (SELECT y
               FROM S y
               WHERE x.A = y.B)
```

```
[ 1 | x ← R,
  [ true | _ ← [ y | y ← S, x.A = y.B ]_{bag} some ]_{sum}

[ 1 | x ← R, [ x.A = y.B | y ← S ]_{some} ]_{sum}
```
Comprehension Unnesting

\[
[ e | qs_1, v \leftarrow [ ]^N, qs_3 ]^M
\]

\[
[ e | qs_1, v \leftarrow [ e_2 ]^N, qs_3 ]^M
\]

\[
[ e | qs_1, v \leftarrow [ e_2 | qs_2 ]^N, qs_3 ]^M
\]

\[
[ e | qs_1, [ e_2 | qs_2 ]^{some}, qs_3 ]^M
\]
Comprehension Unnesting

\[
\begin{align*}
\left[ e \mid qs_1, v \leftarrow [ ]^N, qs_3 \right]^M
\end{align*}
\]

\[
\begin{align*}
\left[ e \mid qs_1, v \leftarrow [ e_2 ]^N, qs_3 \right]^M
\end{align*}
\]

\[
\begin{align*}
\left[ e[e_2/v] \mid qs_1, qs_3[e_2/v] \right]^M
\end{align*}
\]

\[
\begin{align*}
\left[ e \mid qs_1, v \leftarrow [ e_2 \mid qs_2 ]^N, qs_3 \right]^M
\end{align*}
\]

\[
\begin{align*}
\left[ e[e_2/v] \mid qs_1, qs_2, qs_3[e_2/v] \right]^M
\end{align*}
\]

\[
\begin{align*}
\left[ e \mid qs_1, [ e_2 \mid qs_2 ]^{some}, qs_3 \right]^M \quad (\oplus^M \text{idempotent})
\end{align*}
\]

\[
\begin{align*}
\left[ e \mid qs_1, qs_2, e_2, qs_3 \right]^M
\end{align*}
\]
When Syntax Distracts

On Optimizing an SQL-like Nested Query

WON KIM
IBM Research

SQL is a high-level nonprocedural data language which has received wide recognition in relational databases. One of the most interesting features of SQL is the nesting of query blocks to an arbitrary depth. An SQL-like query nested to an arbitrary depth is shown to be composed of five basic types of nesting. Four of them have not been well understood and more work needs to be done to improve their execution efficiency. Algorithms are developed that transform queries involving these basic
An SQL-like query nested to an arbitrary depth is shown to be composed of five basic types of nesting. Four of them have not been well understood.
When Syntax Distracts

On Optimizing an SQL-like Nested Query

WON KIM
IBM Research

An SQL-like query nested to an arbitrary depth is shown to be composed of five basic types of nesting. Four of them have not been well understood.

Implemented in most RDBMSs to this day
When Syntax Distracts

• **Syntactic** classification of nested SQL queries into types $N, Nx, D, J, A, JA, JA(NA), JA(AA), JA(AN), \ldots$

• Classes are associated with their particular SQL–level unnesting rewrites.
When Syntax Distracts

• **Syntactic** classification of nested SQL queries into types $N, N_x, D, J, A, JA, JA(NA), JA(AA), JA(AN), \ldots$

• Classes are associated with their particular SQL–level unnesting rewrites.
When Syntax Distracts

```sql
SELECT DISTINCT f(x)
FROM R AS x
WHERE p(x) IN (SELECT g(y)
    FROM S AS y
    WHERE q(x,y))
```
When Syntax Distracts

```sql
SELECT DISTINCT f(x)
FROM R AS x
WHERE p(x) IN (SELECT g(y)
    FROM S AS y
    WHERE q(x,y))
```

\[
\begin{array}{ll}
[ f(x) ] & x \leftarrow R, \\
[ p(x) = v ] & v \leftarrow [ g(y) ] y \leftarrow S, q(x,y) \end{array}
\]
When Syntax Distracts

```
SELECT DISTINCT f(x)
FROM R AS x
WHERE p(x) IN (SELECT g(y)
                FROM S AS y
                WHERE q(x,y))
```

```
[ f(x) | x ← R,
    [ p(x) = g(y) | y ← S, q(x,y) ]some ]set
```
When Syntax Distracts

SELECT DISTINCT f(x)
FROM R AS x
WHERE p(x) IN (SELECT g(y)
    FROM S AS y
    WHERE q(x,y))

[ f(x) | x ← R, y ← S, q(x,y), p(x) = g(y) ]^{set}

SELECT DISTINCT f(x)
FROM R AS x, S AS y
WHERE q(x,y)
AND p(x) = g(y)
A Zoo of Query Representations

Groupwise Processing of Relational Queries

Damianos Chatziantoniou* Kenneth A. Ross*
Department of Computer Science, Columbia University
damianos,kar@cs.columbia.edu
A Zoo of Query Representations

Groupwise Processing of Relational Queries

Damianos Chatziantoniou*    Kenneth A. Ross*
Department of Computer Science, Columbia University
damianos,kar@cs.columbia.edu

Groupwise Processing of Relational Queries
D. Chatziantoniou, K.A. Ross, VLDB 1997

\[
\text{SELECT } f(x), \text{agg}(g(x)) \\
\text{FROM } R \text{ AS } x \\
\text{GROUP BY } f(x)
\]
A Zoo of Query Representations

Groupwise Processing of Relational Queries

Damianos Chatziantoniou*  Kenneth A. Ross*
Department of Computer Science, Columbia University
damianos,kar@cs.columbia.edu

Groupwise Processing of Relational Queries
D. Chatziantoniou, K.A. Ross, VLDB 1997

\[
\langle f(x), [g(y) \mid y \leftarrow R, f(y) = f(x)]^{agg} \mid x \leftarrow R \rangle^{set}
\]
A Zoo of Query Representations

Groupwise Processing of Relational Queries

Damianos Chatziantoniou* Kenneth A. Ross*
Department of Computer Science, Columbia University
damianos,kar@cs.columbia.edu

Groupwise Processing of Relational Queries
D. Chatziantoniou, K.A. Ross, VLDB 1997

\[ Q f g \text{agg} R \equiv \]

\[ [ \langle f(x), [ g(y) \mid y \leftarrow R, f(y) = f(x) \rangle^{agg} \mid x \leftarrow R ]^{set} ]^{set} \]
A Zoo of Query Representations
A Zoo of Query Representations
A Zoo of Query Representations
A Zoo of Query Representations
A Zoo of Query Representations
A Zoo of Query Representations

```
SELECT agg(g(x))
FROM   P AS x
```
A Zoo of Query Representations

\[ Q' g \text{ agg} P \equiv \left[ g(y) \mid y \leftarrow P \right]^{agg} \]
A Zoo of Query Representations

\[ Q' \ g \ \text{agg} \ P \equiv [\ g(y) \mid y \leftarrow P \ ]^{\text{agg}} \]
A Zoo of Query Representations
what we mean by a group query. We give a syntactic criterion for identifying group queries and prove that this
what we mean by a group query. We give a syntactic criterion for identifying group queries and prove that this

We shall define below the notion of a query graph. A query graph has nodes that are relational operations. We
A Zoo of Query Representations

what we mean by a group query is a syntactic criterion for identifying group of columns. A query graph has nodes that are

We shall define below the notion of a query graph. A

SQL surface syntax, relational algebra, query graphs + annotations, iteration
A Uniform Query Representation
A Uniform Query Representation

\[ Q' \ g \ agg \ P = [ g(y) \mid y \leftarrow P ]^{agg} \]

\[ \text{partition} f \ xs = [ \langle f(x), [ y \mid y \leftarrow xs, f(x) = f(y) ]^M \rangle \mid x \leftarrow xs ]^{set} \]

\[ \text{map} f \ xs = [ f(x) \mid x \leftarrow xs ]^M \]
A Uniform Query Representation

\[ Q' \text{ g agg } P = [ g(y) | y \leftarrow P ]^{agg} \]

\[
\text{partition } f xs = [ \langle f(x) , [ y | y \leftarrow xs , f(x) = f(y) ]^M \rangle | x \leftarrow xs ]^{set}
\]

\[
\text{map } f xs = [ f(x) | x \leftarrow xs ]^M
\]

\[
\text{map } (\lambda \langle x,P \rangle. \langle x , Q' \text{ g agg } P \rangle)(\text{partition } f xs)
\]
A Uniform Query Representation

\[ Q' \; g \; \text{agg} \; P = [ g(y) \mid y \leftarrow P ]^{\text{agg}} \]

\[
\text{partition} \; f \; xs = [ \langle f(x), [ y \mid y \leftarrow xs, f(x) = f(y) ]^M \rangle \mid x \leftarrow xs ]^{set}
\]

\[
\text{map} \; f \; xs = [ f(x) \mid x \leftarrow xs ]^M
\]

\[
\text{map} \; (\lambda \langle x, P \rangle. \langle x, Q' \; g \; \text{agg} \; P \rangle) \; (\text{partition} \; f \; xs)
\]

\[
[ \langle f(x), [ g(y) \mid y \leftarrow R, f(y) = f(x) ]^{agg} \rangle \mid x \leftarrow R ]^{set}
\]
A Uniform Query Representation

$$Q' \ g \ agg \ P = \left[ g(y) \mid y \leftarrow P \right]^{agg}$$

$$\text{partition } f \ xs = \left[ \langle f(x), \left[ y \mid y \leftarrow xs, f(x) = f(y) \right]^{M} \rangle \mid x \leftarrow xs \right]^{set}$$

$$\text{map } f \ xs = \left[ f(x) \mid x \leftarrow xs \right]^{M}$$

$$\text{map } (\lambda \langle x,P \rangle . \langle x, Q' \ g \ agg \ P \rangle ) (\text{partition } f \ xs)$$

SELECT   \ f(x), \ agg(g(x))
FROM      R AS x
GROUP BY  \ f(x)
XPath
XPath Comprehensions

/\texttt{descendant::a}[\texttt{following::b}]/\texttt{child::c}
XPath Comprehensions

/descendant::a[following::b]/child::c

1. Normalize, simplify, **flip** XPath step expressions
XPath Comprehensions

1. Normalize, simplify, **flip** XPath step expressions
2. **Compile** XPath into queries over tabular XML encoding
XPath Comprehensions

\[ /\text{descendant}::a[\text{following}::b]/\text{child}::c \]

1. Normalize, simplify, **flip** XPath step expressions

2. **Compile** XPath into queries over tabular XML encoding

\[
\text{xpath}(p) c = \text{xpath } p \text{ (root } c) \\
\text{xpath}(p_1/\neg p_2) c = [ n' | n \leftarrow \text{xpath } p_1 c, n' \leftarrow \text{xpath } p_2 n ]^X \\
\text{xpath}(p [q]) c = [ n | n \leftarrow \text{xpath } p c, [ \text{true} | _ \leftarrow \text{xpath } q n ]^{\text{some}} ]^X \\
\text{xpath}(ax::t) c = \text{step } (ax::t) c
\]
A Tabular XML Encoding

```xml
<a>
  <b><c><d/>e</c></b>
  <f><!--g-->  
    <h><i/></h>
  </f>
</a>
```
A Tabular XML Encoding
A Tabular XML Encoding
A Tabular XML Encoding
A Tabular XML Encoding
A Tabular XML Encoding

\[
\text{step (descendant:::t) } c = \\
\left[ n \mid n \leftarrow \text{doc}, \ \text{pre } c < \text{pre } n, \ \text{post } c > \text{post } n, \ \text{tag } n = t \right]^X
\]
A Tabular XML Encoding

step (descendant :: t) c =
[ n | n ← doc, pre c < pre n, post c > post n, tag n = t ]^x
A Tabular XML Encoding

\[
\text{step} (\text{descendant}::t) c = \\
\left[ n \mid n \leftarrow \text{doc}, \ \text{pre} c < \text{pre} n, \ \text{post} c > \text{post} n, \ \text{tag} n = t \right]^X
\]

\[
\text{step} (\text{ancestor}::t) c = \\
\left[ n \mid n \leftarrow \text{doc}, \ \text{pre} c > \text{pre} n, \ \text{post} c < \text{post} n, \ \text{tag} n = t \right]^X
\]
XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

Institute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002
XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

Institute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002
XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

Institute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002

```
/descendant::*[name()='g']/preceding::*[name()='c']
```
XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

Institute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002

/descendant::*:g/preceding::*:c

Diagram:

- Root node 'a'
- Child node 'b'
- Child node 'c'
- Sub-child node 'd'
- Sub-child node 'e'
- Child node 'g'
- Sub-child node 'i'
- Child node 'h'
- Child node 'j'

Arrows indicating the path from 'g' to 'c'.
XPath: Looking Forward

Dan Olteanu, Holger Meuss, Tim Furche, François Bry

Institute for Computer Science and Center for Information and Language Processing
University of Munich, Germany

XPath: Looking Forward
D. Olteanu et al., XMLDM (EDBT 2002), March 2002

```
/ancestor::g/preceding::c
```

```
/ancestor::c/following::g
```

```
/ancestor::c[c[following::g]]
```

```
/ancestor::c/c[following::g]
```
XPath: Looking Forward
XPath: Looking Forward

\[ p/descendant::n[preceeding::m] \equiv p[preceeding::m]/descendant::n \]
\[ | p/child::*[descendant-or-self::m] \]
\[ | following-sibling::*[descendant-or-self::m] \]

\[ /descendant::n[preceeding::m] \equiv /descendant::m/following::n \]

\[ p/child::n[preceeding::m] \equiv p[preceeding::m]/child::n \]
\[ | p/child::*[descendant-or-self::m] \]
\[ | following-sibling::n \]

\[ p/self::n[preceeding::m] \equiv p[preceeding::m]/self::n \]

\[ p/following-sibling::n[preceeding::m] \equiv p[preceeding::m]/following-sibling::n \]
\[ | p/following-sibling::*[descendant-or-self::m] \]
\[ | following-sibling::n \]
\[ | p[descendant-or-self::m]/following-sibling::n \]

\[ p/following::n[preceeding::m] \equiv p[preceeding::m]/following::n \]
\[ | p/following::m/following::n \]
\[ | p[descendant-or-self::m]/following::n \]
Comprehending XPATH
Comprehending XPath

/ descendant::g/ preceding:::c
Comprehending XPath

\[
\text{/descendant::g/preceding::c}
\]

\[
[ v' \mid v \leftarrow \text{doc}, \text{tag } v = 'g', v' \leftarrow \text{doc}, \text{pre } v' < \text{pre } v, \text{post } v' < \text{post } v, \text{tag } v' = 'c' ]^X
\]
Comprehending XPath

/\text{descendant::g/preceding::c}

\[
\left[ v' \mid v \leftarrow \text{doc}, \quad \text{tag } v = 'g', \quad v' \leftarrow \text{doc}, \quad \text{pre } v' < \text{pre } v, \quad \text{post } v' < \text{post } v, \quad \text{tag } v' = 'c' \right]^X
\]

/\text{descendant::c[following::g]}
Comprehending XPath

\[
/\text{descendant::*:g/preceding::*:c}
\]

\[
[ v' | v \leftarrow \text{doc}, \quad \text{tag } v = 'g', \quad v' \leftarrow \text{doc}, \quad \text{pre } v' < \text{pre } v, \quad \text{post } v' < \text{post } v, \quad \text{tag } v' = 'c' ]^X
\]

SELECT DISTINCT v'  
FROM doc v, doc v'  
WHERE tag v = 'g' AND tag v' = 'c'  
AND pre v' < pre v AND post v' < post v  
ORDER BY pre v'
BRING BACK MONAD COMPREHENSIONS
Comprehensions in Haskell
Comprehensions in Haskell
Comprehensions in Haskell

Comprehending Monads


Torsten Grust
U Tübingen
Comprehensions in Haskell

Comprehending Monads

Comprehensions in Haskell

Comprehending Monads

Haskell 1.4
Comprehensions in Haskell

Comprehending Monads


Haskell 1.4

Haskell 98

Torsten Grust
Comprehensions in GHC

Comprehending Monads


Haskell 1.4
Haskell 98

Torsten Grust

U Tübingen
Comprehensions in GHC

Comprehending Monads

Haskell 1.4

Haskell 98

Comprehensive Comprehensions

Comprehensions in GHC

- Comprehending Monads
- Haskell 1.4
- Haskell 98
- Bringing Back Monad Comprehensions
- Comprehensive Comprehensions

Timeline:
- 1990
- 1997
- 2003
- 2007
- 2011

Torsten Grust

U Tübingen
Movie Plot Line

Meeting

Inciting Incident

Pinch

Climax

Turning Point

5 MIN

120 MIN
Comprehensive Comprehensions
Comprehensions with ‘Order by’ and ‘Group by’

Philip Wadler
University of Edinburgh

Simon Peyton Jones
Microsoft Research

Comprehensive Comprehensions
P. Wadler, S. Peyton-Jones, Haskell Workshop, October 2007
\[
\begin{array}{l}
\left( \text{the dept, maximum salary} \right) \\
\left( \text{name, dept, salary} \right) \leftarrow \text{employees} \\
\text{then group by dept using groupWith} \\
\text{length dept > 10} \\
\text{then sortWith by Down (sum salary)} \\
\text{then take 5}
\end{array}
\]
Comprehensive Comprehensions

Comprehensions with ‘Order by’ and ‘Group by’

Philip Wadler
University of Edinburgh

Simon Peyton-Jones
Microsoft Research

[ (the dept, maximum salary)
  | (name, dept, salary) <- employees
  , then group by dept using groupWith
  , length dept > 10
  , then sortWith by Down (sum salary)
  , then take 5
  ]
Comprehensive Comprehensions

Comprehensions with ‘Order by’ and ‘Group by’

Philip Wadler
University of Edinburgh

Simon Peyton Jones
Microsoft Research

Comprehensive Comprehensions
P. Wadler, S. Peyton-Jones, Haskell Workshop, October 2007

Not shown: set operations, joins, WITH RECURSIVE, ...

[ sum salary
| (name, "MS", salary) <- employees
, then group using runs 3
, then take 5
]
Database–Supported Haskell

Haskell Heap

DBMS

DATA
Database–Supported Haskell

Haskell Heap

DBMS
Database–Supported Haskell

- DATA
- Haskell Heap
- Haskell
- DBMS

Torsten Grust
Database–Supported Haskell

Haskell

Haskell Heap

DATA

DBMS
Database–Supported Haskell

Haskell Heap

SQL

DATA

DBMS
Database–Supported Haskell

Haskell Heap

SQL

DATA

DBMS
Database–Supported Haskell

Haskell Heap

DBMS

DATA

SQL

SQL
Database–Supported Haskell

Haskell Heap

DATA

SQL

DBMS
Database–Supported Haskell
-- rolling minimum (mins [3,4,1,7] = [3,3,1,1])
mins :: Ord a => Q [a] -> Q [a]
mins xs =
    [ minimum [ y | (y,j) <- #xs, j <= i ] | (_,i) <- #xs ]

-- margin: current value - minimum value up to now
margins :: (Ord a, Num a) => Q [a] -> Q [a]
margins xs = [ x - y | (x,y) <- zip xs (mins xs) ]

-- our profit is the maximum margin obtainable
profit :: (Ord a, Num a) => Q [a] -> Q [a]
profit xs = maximum (margins xs)

-- best profit obtainable for stock on given date
bestProfit :: Text -> Date -> Q [Trade] -> Q Double
bestProfit stock date trades =
    profit [ price t | t <- sortWith ts trades, id t == stock, day t == date ]
-- rolling minimum (mins [3,4,1,7] = [3,3,1,1])
mins :: Ord a => [a] -> [a]
mins xs =
    [ minimum [ y | (y,j) <- #xs, j <= i ] | (_,i) <- #xs ]

-- margin: current value - minimum value up to now
margins :: (Ord a, Num a) => [a] -> [a]
margins xs = [ x - y | (x,y) <- zip xs (mins xs) ]

-- our profit is the maximum margin obtainable
profit :: (Ord a, Num a) => [a] -> [a]
profit xs = maximum (margins xs)

-- best profit obtainable for stock on given date
bestProfit :: Text -> Date -> [Trade] -> Double
bestProfit stock date trades =
    profit [ price t | t <- sortWith ts trades, id t == stock, day t == date ]
Database–Supported Haskell

-- rolling minimum (mins [3,4,1,7] = [3,3,1,1])
mins :: Ord a => Q [a] -> Q [a]
mins xs =
  [ minimum [ y | (y,j) <- #xs, j <= i ] | (_,i) <- #xs ]

-- margin: current value - minimum value up to now
margins :: (Ord a, Num a) => Q [a] -> Q [a]
margins xs = [ x - y | (x,y) <- zip xs (mins xs) ]

-- our profit is the maximum margin obtainable
profit :: (Ord a, Num a) => Q [a] -> Q [a]
profit xs = maximum (margins xs)

-- best profit obtainable for stock on given date
bestProfit :: Text -> Date -> Q [Trade] -> Q Double
bestProfit stock date trades =
  profit [ price t | t <- sortWith ts trades,
                id t == stock, day t == date ]
-- rolling minimum (mins [3,4,1,7] = [3,3,1,1])
mins :: Ord a => [a] -> [a]
mins xs =
    [ minimum [ y | (y,j) <- #xs, j <= i ] | (_,i) <- #xs ]

-- margin: current value - minimum value up to now
margins :: (Ord a, Num a) => [a] -> [a]
margins xs = [ x - y | (x,y) <- zip xs (mins xs) ]

-- our profit is the maximum margin obtainable
profit :: (Ord a, Num a) => [a] -> [a]
profit xs = maximum (margins xs)

-- best profit obtainable for stock on given date
bestProfit :: Text -> Date -> [Trade] -> Double
bestProfit stock date trades =
    profit [ price t | t <- sortWith ts trades,
               id t == stock, day t == date ]
-- SQL code generated from Haskell source
SELECT MAX(margins.price - margins.min)
FROM
(SELECT t.price,
    MIN(t.price)
    OVER (ORDER BY t.ts ROW BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW)
FROM trades AS t
WHERE t.id = 'ACME'
AND t.day = '07/01/2015'
) AS margins(price,min)
Comprehensions
Yield Independent Work
Comprehensions
Yield Independent Work

\[
\left[ \left[ f y \mid y \leftarrow g x \right] \mid x \leftarrow xs \right]
\]

\[ f :: a \rightarrow b \]
Comprehensions Yield Independent Work

\[
\left[ f \left[ y \mid y \leftarrow g \ x \right] \mid x \leftarrow xs \right]
\]

\[ f :: a \rightarrow b \]
Comprehensions
Yield Independent Work

\[
[f^1 [\ y \mid y \leftarrow g \ x \] \mid x \leftarrow xs ]
\]

\[
f :: a \rightarrow b
\]

\[
f^1 :: [a] \rightarrow [b]
\]
Comprehensions
Yield Independent Work

\[
\begin{align*}
\forall y \left(y \leftarrow g \, x \right) \mid x \leftarrow x s \\
\end{align*}
\]

\[
\begin{align*}
f^2 :: [a] \rightarrow [b] \\
f^1 :: [a] \rightarrow [b] \\
f^2 :: [[a]] \rightarrow [[b]]
\end{align*}
\]
Comprehensions
Yield Independent Work

\[ f^2 \left[ g \ x \ | \ x \leftarrow xs \right] \]

\[
\begin{align*}
f &:: a \rightarrow b \\
f^1 &:: [a] \rightarrow [b] \\
f^2 &:: [[a]] \rightarrow [[b]]
\end{align*}
\]
Comprehensions
Yield Independent Work

\[ f^2 (g^1 \, xs) \]

\[ f :: a \rightarrow b \]
\[ f^1 :: [a] \rightarrow [b] \]
\[ f^2 :: [[a]] \rightarrow [[b]] \]
Comprehensions
Yield Independent Work

\[ [f^n e \mid x \leftarrow xs] \leadsto f^{n+1} [e \mid x \leftarrow xs] \]
Nested Data Parallelism

\[ f^n e \mid x \leftarrow xs \] \implies f^{n+1} \[ e \mid x \leftarrow xs \]

Implementation of a Portable Nested Data-Parallel Language*

Guy E. Blelloch\(^1\)  Siddhartha Chatterjee\(^2\)
Jonathan C. Hardwick  Jay Sipelstein  Marco Zagha

*Implementation of a Portable Nested Data-Parallel Language
G. E. Blelloch et al., ACM PPoPP, May 1993
The Flatter, the Better
The Flatter, the Better

\[ xss + 2 yss \]
The Flatter, the Better

\[ xss +^2 yss \]

\[
\begin{bmatrix}
[19, 0], [30], [11, 10, 7]
\end{bmatrix}
\]

\[
\begin{bmatrix}
[0, 4], [12], [13, 2, 3]
\end{bmatrix}
\]
The Flatter, the Better

\[ \text{\textit{xss} } +^2 \text{\textit{yss}} \]

\[
\begin{bmatrix}
[19, 0], [30], [11, 10, 7] \\
\end{bmatrix}
\]

\[
\begin{bmatrix}
[0, 4], [12], [13, 2, 3]
\end{bmatrix}
\]
The Flatter, the Better

\[ xss +^2 yss \]

\[
\begin{bmatrix}
19, 0, 30, 11, 10, 7 \\
0, 4, 12, 13, 2, 3
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]
The Flatter, the Better

\[ xss +^2 yss \]

\[
\begin{bmatrix}
19, 4, 42, 24, 12, 10
\end{bmatrix}
\]

\[
\begin{bmatrix}
\end{bmatrix}
\]
The Flatter, the Better

\[ xss +^2 yss \]

\[ [[19, 4], [42], [24, 12, 10]] \]
The Flatter, the Better

\[ f^n e \rightsquigarrow \text{imprint}_{n-1}(f^1(\text{forget}_{n-1} e)) \]

\[[[19, 4], [42], [24, 12, 10]]\]
The Flatter, the Better

\[ f^n e \mapsto \text{imprint}_{n-1}(f^1(\text{forget}_{n-1} e)) \]

\[
[[19, 4], [42], [24, 12, 10]]
\]
The Flatter, the Better

\[ f^n e \rightsquigarrow \text{imprint}_{n-1}(f^1(\text{forget}_{n-1} e)) \]

\[
[[19, 4], [42], [24, 12, 10]]
\]
The Flatter, the Better

$$f^n e \sim \text{imprint}_{n-1}(f^1(\text{forget}_{n-1} e))$$

$$[[19, 4], [42], [24, 12, 10]]$$
The Flatter, the Better

\[ f^n e \rightleftharpoons \text{imprint}_{n-1}(f^1(\text{forget}_{n-1} e)) \]

\[[[19, 4], [42], [24, 12, 10]]\]
Database Systems: Designed to Implement
Database Systems: Designed to Implement $1$
Database Systems: Designed to Implement

\[ \text{seg} \quad \ldots \quad x \quad y \]

<p>| | | |</p>
<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>
Database Systems: Designed to Implement

\[ \pi_{\text{sum}: x+y}(\{ 1 \}) \]

\[
\begin{array}{c|cc}
\text{seg} & x & y \\
1 & 19 & 0 \\
1 & 0 & 4 \\
2 & 30 & 12 \\
3 & 11 & 13 \\
3 & 0 & 2 \\
3 & 7 & 3 \\
\end{array}
\]
Database Systems:
Designed to Implement

\[ \sum \]

\[ \pi_{\text{sum: } x+y}() \]

\[
\begin{array}{c|cc}
\text{seg} & x & y \\
\hline
1 & 19 & 0 \\
1 & 0 & 4 \\
2 & 30 & 12 \\
3 & 11 & 13 \\
3 & 0 & 2 \\
3 & 7 & 3 \\
\end{array}
\]
\[ +^1 \]

\[ \pi_{\text{sum}: x+y}( \quad ) \]

\[ \times_{p}^1 \]
Database Systems: Designed to Implement

\[ \pi_{\text{sum}: x+y} \left( \sum_{i=1}^{1} \right) \]

\[ (1, 19, 0) \]
\[ (1, 0, 4) \]
\[ (2, 30, 12) \]
\[ (3, 11, 13) \]
\[ (3, 0, 2) \]
\[ (3, 7, 3) \]

\[ \Delta_{p} \]

\[ \sum_{i=1}^{1} \]

\[ \times_{p} \]

\[ \pi_{\text{sum}: x+y} \left( \sum_{i=1}^{1} \right) \]

\[ (1, 19, 0) \]
\[ (1, 0, 4) \]
\[ (2, 30, 12) \]
\[ (3, 11, 13) \]
\[ (3, 0, 2) \]
\[ (3, 7, 3) \]
Database Systems: Designed to Implement

\[ \pi_{\text{sum: } x+y}(\mathbf{x} + \mathbf{y}) \]

\[ \mathbf{x} \times_p \mathbf{y} = \mathbf{z} \]

\[ x_1, \ldots, x_n \]
Database Systems: Designed to Implement

\[ +^1 \]

\[ \pi_{\text{sum}: x+y}( \downarrow ) \]

\[ \times_{p^1} \]

\[ \times_p \land \text{seg}_1 = \text{seg}_2 \]

<table>
<thead>
<tr>
<th>seg</th>
<th>x</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
<td>3</td>
</tr>
</tbody>
</table>

\[ \text{seg}_1 \]

<table>
<thead>
<tr>
<th>seg</th>
<th>x</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>30</td>
</tr>
<tr>
<td>3</td>
<td>11</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>7</td>
</tr>
</tbody>
</table>

\[ \text{seg}_2 \]

<table>
<thead>
<tr>
<th>seg</th>
<th>y</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>0</td>
</tr>
<tr>
<td>1</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>13</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
</tr>
</tbody>
</table>
Plan Bundles
Instead of Query Avalanches

\((\text{Int}, [\text{Str}], [[[\text{Bool}, [\text{Int}, \text{Int}]]]])\)
Plan Bundles
Instead of Query Avalanches

[(Int, [Str]), [(Bool, [Int, Int])]]
Plan Bundles Instead of Query Avalanches
Plan Bundles Instead of Query Avalanches

Query Plan Bundle

[ ] [ ] [ ] [ ]
Plan Bundles
Instead of Query Avalanches

Query Plan Bundle

Torsten Grust
U Tübingen
There was NO WARNING of their ARRIVAL!
There was NO LIMIT to their NESTING!